EG-ARES-06-5

GNC Architecture Design for ARES Simulation

Architecture for Exploration Studies (ARES)
DES

Revision 3.0
26 June 2006

National Aeronautics and
Space Administration

Lyndon B. Johnson Space Center
Houston, Texas

DOCUMENT NUMBER	**EG-ARES-06-5**
DOCUMENT DATE	**26 June 2006**
TITLE	**GNC Architecture Design for ARES Simulation**
SYNOPSIS	**This document describes the GNC architecture design for all ARES simulations.**

Prepared by

Robert S. Gay
Aeroscience and Flight Mechanics Division
Engineering Directorate

CHANGE LOG

Revision	Date	Originator	Description of change
1.0	18 Dec 2005	R. Gay	Draft - Baseline Release
1.0	09 Jan 2006	R. Gay	Baseline Release
1.1	09 Jan 2006	R. Gay	Made sim comfig one init job since it uses pointers
1.2	09 Jan 2006	R. Gay	Use trick input processor to assign sim config pointers. AFM replaced by OAE
3.0	23 June 2006	R. Gay	Version 2.0 reflected "as-coded" version of 1.2 (see ARES_GNC_arch_overview_2.0.ppt). for details. Version 2.1.1 reflected moving the gnc_executive to veh_executive (see ARES_GNC_arch_overview_2.1.1.ppt). Version 3.0 reflects updates including: "previous" task data option, "previous" sub-mode option, and moving all mode transition logic to the veh_executive. A new GNC state and status data structure was also added to facilitate mode transition logic. See ARES_GNC_arch_update_to_3.0.ppt for update details and see ARES_GNC_arch_overview_3.0.ppt for summary overview.

TABLE OF CONTENTS

APPENDICES

FIGURES

PREFACE

This document describes the GNC architecture design and associated interfaces required for all ARES simulations. Questions about this document should be addressed to Robert Gay (robert.gay-1@nasa.gov, 281-483-6330).

1.0 INTRODUCTION

The purpose of this document is to describe the GNC architecture and associated interfaces for all ARES simulations. Establishing a common architecture facilitates development across the ARES simulations and provides an efficient mechanism for creating an end-to-end simulation capability. In general, the GNC architecture is the frame work in which all GNC development takes place, including sensor and effector models. All GNC software applications have a standard location within the architecture making integration easier and, thus more efficient.

2.0 GENERAL REQUIREMENTS

2.1 GNC architecture shall handle all flight phases.

2.2 GNC architecture shall be modular and scalable.

2.3 GNC architecture shall support the capability to bring together independently developed "major" flight phases, such as ascent, orbit, entry, etc., in an end-to-end simulation.

2.4 GNC architecture shall allow for running multiple modules (sensors, navigation, guidance, control/jet selection, and effectors) simultaneously, while closing the loop on a single string.

2.4.1 GNC architecture shall allow for single-string versions of the simulation.

3.0 GNC ARCHITECTURE DESCRIPTION

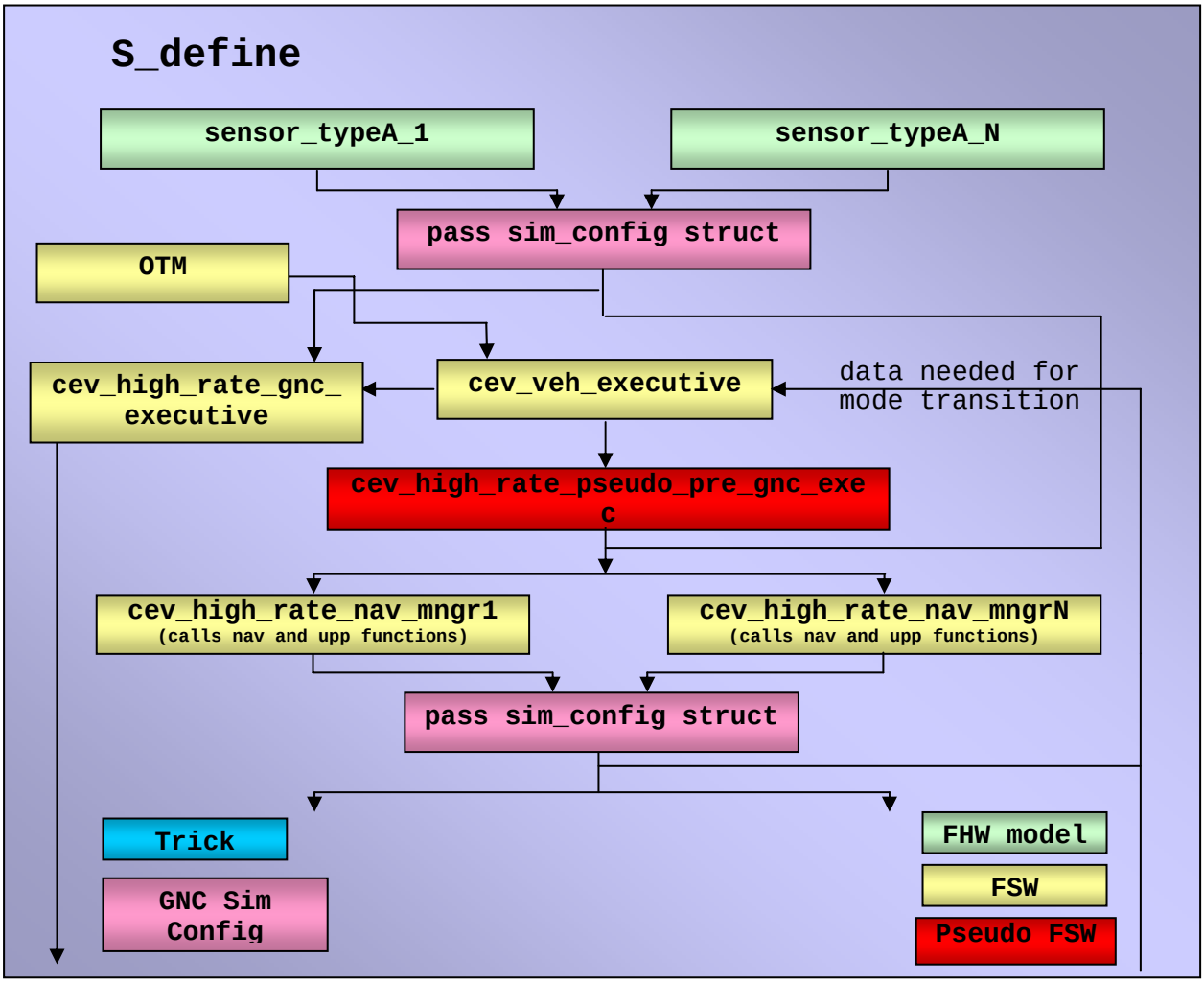
3.1 S_define-Level Model Flow Diagrams

The following flow diagrams shown in

Figure 1, demonstrate the GNC architecture by illustrating the function calls and data flow at the S_define level for the trick-based ARES simulations. The naming convention shown in the diagrams should be followed in all ARES simulation development; however, descriptors may be used for each type of manager function such as "cev_high_rate_pseudo_nav_mngr" etc.

3.1.1 SIM_xyx (single rate group) Flow Diagram

Figure 1 shows the GNC architecture for all ARES simulations with a single (HIGH) rate group.



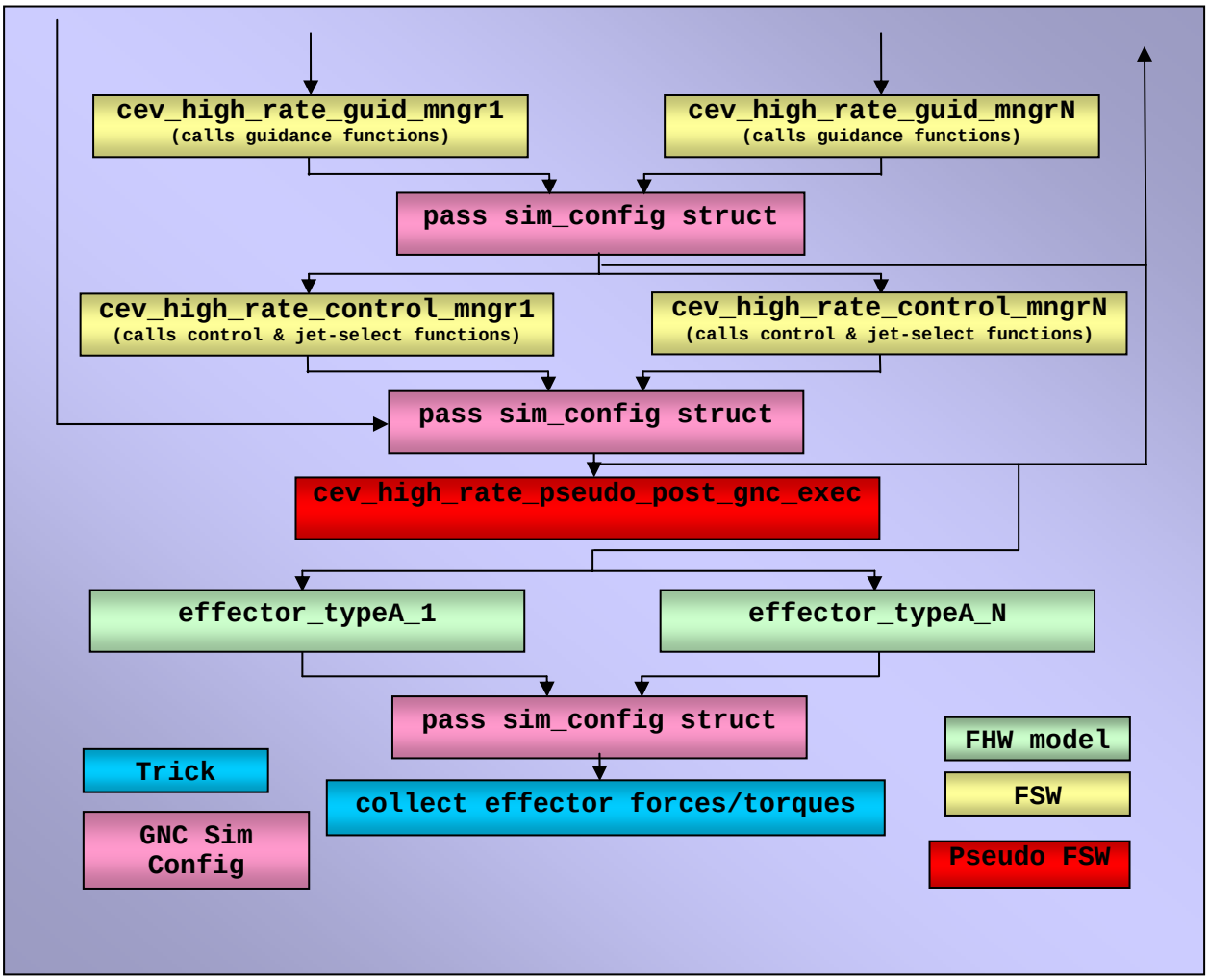


Figure 1: SIM_xyz GNC Architecture Flow Diagram (single rate group)

3.2 GNC Simulation Configuration

In order to allow for multiple versions of sensors, navigation, guidance, control, and effectors to be active simultaneously, while closing-the-loop on a single string, a non-flight software simulation configuration data structure is necessary. This structure is used to take all applicable outputs and pass (via a pointer) only the desired data down the GNC chain: sensors – navigation – guidance – control – effectors. GNC state and status information is also passed with sim_config parameters back to the vehicle executive for mode transitions. Since pointers are used, the memory addresses only need to be assigned once at initialization. Fortunately, trick allows this to be done with input data using standard "C" syntax. Figure 2 contains an example of the GNC simulation configuration data structure.

```
typedef struct{

   cev_gnc_guid_ss_T    * guid_ss;   /* --  Guidance state and status  */
   cev_gnc_nav_ss_T     * nav_ss;    /* --  Nav state and status       */
   cev_gnc_cntrl_ss_T   * cntrl_ss;  /* --  Control state and status   */
   pose_out_T           * LDO;       /* --  LIDAR sensor output       */
   pose_out_T           * NFO;       /* --  NFIR output               */
   imu_out_T            * IMUO;      /* --  IMU output                */
   gps_out_T            * GPSO;      /* --  GPS output                */
   str_out_T            * STRO;      /* --  Star trckr out            */
   general_nav_T        * NAV;       /* --  General nav struct        */
   general_guid_out_T   * GO;        /* --  General guid output struct */
   CONTRL_PARAM         * CTRL;      /* --  General control struct     */
   double             eff_force[3];  /* N   effector force            */
   double             eff_torque[3]; /* NM  effector torque           */

} veh_gnc_sim_config_T;
```

Figure 2: GNC Simulation Configuration Data Structure Example

3.3 Rate Groups

Sensor models, effector models, and the Onboard Trajectory Manager (OTM) run at whatever frequency is desired for the given function, where as the Vehicle Executive, GNC Executive (or pseudo GNC executives), and the Manager functions are divided into appropriate rate groups: LOW, MEDIUM, and HIGH for example. The Vehicle Executive is always called at the "HIGH" rate. The GNC executive (or pseudo GNC executives) and the Manager function names are prefixed with "low_rate", "med_rate", or "high_rate" descriptors to designate their frequency. Clearly, all the functions executed within a Manager are called at the same rate, unless additional logic is added to slow down a given function call.

3.4 Sensor Models

The purpose of sensor models is to emulate the functionality of sensor flight hardware. Each model will be called at the S_define level near the top of the vehicle sim_object (see

Figure 1).

3.5 Phase, Segment, Mode, Sub-modes, and Tasks

Phase: Major flight regime: Ascent, Orbit, Entry, Transit, …

Segment: Logical division of a **Phase**: Far-Field Rendezvous, Near-Field Rendezvous, Prox-ops, Final Approach…(Orbit Phase)

Mode: Definition of general GNC capability: absolute navigation, relative navigation, ascent guidance, Lambert targeting, entry guidance, entry control, ascent control, etc. Each mode is defined by a combination of six types of sub-modes.

Sub-mode: Low-level definition of GNC capability. There are six types of sub-modes: translational and rotational Guidance, Navigation, and Control. Logic in the GNC Manager functions will make decisions based on the current sub-mode.

Task: Series of actions necessary to complete a **Segment**. Each task is an instantiation of a **Mode**, usually involving task-specific data.

Utilizing the Phase, Segment, Mode, and Sub-mode concept, allows the simulation scenario to be largely data driven. Nominal and contingent sequencing can all be predefined if desired and driven by time, events, OTM, crew, ground, etc. See /vobs/cev/sims/xyz_sim/-Modified_data/task_list/*.d, for examples of this type of data.

3.6 Vehicle Executive

The primary purpose of the Vehicle Executive flight software function is to update the Phase, Segment, and Mode if the current Task is complete or an override command is given. Task-specific data is also updated in this function. Furthermore, the Vehicle Executive contains all Mode-transition logic and increments and resets a task-timer, and maintains “previous” Phase, Segment, Mode, and Task parameters. Appendix A contains a list of the commands used to manipulate nominal and contingency sequencing within the Vehicle Executive.

The executive data defines the overall mission scenario. This data also defines the Mode via the associated Sub-modes. The definition of these Sub-modes are determined through code in the Manger functions and task-specific data. S_define file location: Called after the OTM functions in the vehicle sim_object (see

Figure 1).

3.7 GNC Executive (Pseudo GNC Executives)

Currently, the "FSW" or prototype FSW GNC Executive is NOT implemented in ARES. The purpose of this function will not be fully defined in this document. The iGNC Mode Team will produce a formal document defining the GNC Executive in the near future.

The Pseudo Pre GNC Executive and the Pseudo Post GNC Executive are used to simulate the functionality of the FSW GNC Executive, while maintaining all the capabilities of the GNC simulation configuration. Presently, the Pseudo Pre GNC Executive only sets the current Sub-mode. In the future, this function may be responsible for reading data from a "common data area" to be passed to the appropriate GNC Manager functions. The Pseudo Post GNC Executive is, for now, just a place-holder (does nothing) for future development, such as performing data writes to a "common data area".

3.8 Navigation Manager

The Navigation Manager is flight software responsible for configuring and selecting the desired navigation during flight, including sensor input collection and navigation-derived parameter outputs. Decisions within the code are made based on Sub-mode and other necessary parameters with respect to those Sub-modes. Figure 3 contains example pseudo-code for a Navigation Manager. S_define file location: Called after the GNC Executive (or pseudo pre GNC executive) in the vehicle sim_object (see

Figure 1).

```
/* Switch on trans nav sub-mode */
   switch (VEH->t_nav_mode){

   case veh_TN_ORBIT_ABS_ONLY:

      /* Process sensor inputs */
      nav_input(G, time, LO, NFO, IRCO, IMUO, GPSO, STO, TLMO);

      /* Call Hifi nav function */
      navigated_rnp(time, PC->omega, N);
      hifi_nav(time, NAV, G, HFN);

      /*--- Calculate Chaser States ---*/

      /* Call inertial nav functions */
      N->inertial_nav = On;
      N->att_mat      = B_TO_I;
      N->sensed_accel = INERTIAL_SENSED;
      MtxV(body_rate_chsr, chsr_att->T_body2eci, chsr_att->body_rate);
      inrtl_nav_params(&(HFN->filter.state.m[SI->chaser_abs_x]),
                       &(HFN->filter.state.m[SI->chaser_abs_xd]),
                       P->accel.cg_sensed_accel_eci_estimate,
                       chsr_att->T_body2eci, body_rate_chsr, N);

      orb_car_to_elem(N->inertial_pos, N->inertial_vel, PC->mu, C_OE);

      /* Call planet-relative nav functions */
      M_TRANS(T_inertial_body_chsr, N->T_body_inertial);
      planet_relative_state(PRN, N->inertial_pos, N->inertial_vel,
                            T_inertial_body_chsr, N->T_inrtl_pfixed, PC->r_eq,
                            PC->e2, PC->omega);
      break;
   }
```

Figure 3: Navigation Manager Pseudo-Code

3.9 Guidance Manager

The Guidance Manager is flight software responsible for configuring and selecting the desired position, velocity, attitude, and attitude rate guidance for all flight phases. Decision making is based on Sub-modes, and other necessary parameters with respect to those Sub-modes. Figure 4 contains example pseudo-code for a Guidance Manager. S_define file location: Called after the Navigation Manger in the vehicle sim_object (see

Figure 1).

```
/* Switch on translational guidance sub-mode */
   switch (VEH->t_guid_mode){

   case veh_TG_CW:

       if(task_init)
       {
          TGD->POCW.des_met_arrival = time + TGD->POCW.dt_transfer
       }

       omega = -TN->orb_rate_lvlh[1];
       dt_transfer = TGD->POCW.des_met_arrival - time;
       if (dt_transfer < TGD->POCW.stationkeep_dt )
       {
           dt_transfer = TGD->POCW.stationkeep_dt;
       }

       for(i=0;i<3;i++) svi[i+1] = RN->R_tcom_ccom_tlvlh[i];
       for(i=0;i<3;i++) svi[i+4] = RN->V_tcom_ccom_tlvlh_tlvch[i];

       alt_cwtgtng( TGD->POCW.cwTgtngTypeIP,
                    TGD->POCW.cwTgtngTypeOOP
                    omega, dt_transfer, svi,
                    TGD->POCW.des_inplane_pos,
                    TGD->POCW.des_ooplane_pos,
                    tgtng_sing,  dv_reqd_lvlh );

       MtxV(GO->cmnd_delta_v,RN->T_cbody_tlvlh,dv_reqd_lvlh);

       break;
    }
```

Figure 4: Guidance Manager Pseudo-Code

3.10 Control Manager

The Control Manager is flight software responsible for configuring and selecting the desired control, command saturation, and jet selection for all flight phases. Decision making is based on Sub-modes, and other necessary parameters with respect to those Sub-modes. Figure 5 contains example pseudo-code for a Control Manager. S_define file location: Called after the Guidance Manger in the vehicle sim_object (see

Figure 1).

```
/* Switch on att/rate control sub-mode */

   switch (VEH->a_cntrl_mode){

   case cev_AC_RATE_DAMP:
      break;

   case cev_AC_ENTRY_SIMPLE_PD:

      /* Turn on Attitude control modules */
      RCM->Enable_rcs_modulator = On;
      PD->Enable_simple_pd_att_con = On;
      /* Rot and trans mode forces firing of couples */
      SXCM->mode = SIMPLEX_ROT_ONLY;

      /* set RCS config to command module */
      rcs_config = CMRCS;

      V_COPY(PD->att_gain, ACD->kp);
      V_COPY(PD->rate_gain, ACD->kd);

      break;
   }
```

Figure 5: Control Manager Pseudo-Code

3.11 Effector Models

The purpose of effector models is to emulate the functionality of effector flight hardware. Each model will be called at the S_define level after the Control Manager in the vehicle sim_object (see

Figure 1).

3.12 Trick Collect Mechanism Utilization

One of the many advantages of using the trick simulation development environment is the Collect Mechanism. The Collect Mechanism is an abstract data collection utility that allows an arbitrary number of like data types (not required to be the same type, but it makes things easier) to be accumulated at the S_define level and then accessed as desired within any function. This trick capability has many uses, but the most common ones are collecting forces, torques, and gravitational acceleration. In addition, the Orbital_body code utilizes this mechanism to collect all the forces and torques for each body to sum everything for the composite body.

Dynamic collect statements are typically located after each vehicle sim_object. The Orbital_body collect statements are placed after the “dynamics” sim_object. The effector forces and torques to be collected will be the ones passed on by the effector model(see

Figure 1).

3.12.1 Transmitted and Non-Transmitted Forces and Torques

Since the forces and torques for all attached bodies must be accumulated for the composite body, the concept of transmitted and non-transmitted forces and torques was created within the Orbital_body code. If the forces and torques for a given body are to be accumulated for the composite body, then they are denoted as “transmitted”. If however the forces and torques are only to be applied to the body if it is not attached to anything or if it is the “root” (top-level) body, then they are denoted as “non-transmitted”. An example of a transmitted force or torque would be one coming from a jet or control surface, while forces from a body-specific aero model or torques from a body-specific gravity-gradient model would be non-transmitted. Note, non-transmitted forces and torques are imparted on the composite body by computing them with respect to the “root” (top-level) body states. Figure 6 contains examples from the ARES ascent simulation S_define file las sim_object.

```
 /* COLLECT TRANSMITTED EFFECTOR FORCES FOR VEHICLE */
collect las.body.orbit.trans.collect_effector_forc =
{
   las.las_prop.thrust[0]
};

/* COLLECT TRANSMITTED EFFECTOR TORQUES FOR VEHICLE */
collect las.body.orbit.rot.collect_effector_torq =
{
   las.las_prop.moment[0]
};

/* COLLECT DYNAMIC ACCELERATIONS FOR SYSTEM */
collect las.body.orbit.trans.collect_dynamic_accel =
{
   las.grav.out.accel[0],
   las.grav.out.accel_3rd[0]
};

/* COLLECT NON-TRANSMITTED TORQUES FOR VEHICLE */
/* Aero forces and torques are not transmitted to composite vehicle,
   since a single aero model in the cev_cm_veh provides composite aero
   when connected, so aero is only used after las sep to model the
   drag on the tower for separation analyses */
collect las.body.orbit.rot.collect_no_xmit_torq =
{
   las.aero_current.moment_b[0]
};

collect las.body.orbit.trans.collect_no_xmit_forc =
{
   las.aero_current.force_b[0]
};
```

Figure 6: Trick Collect Statements Example

4.0 INTERFACE REQUIREMENTS

In order for the GNC architecture to function, all of its components and the lower-level models used within it must follow a minimum set of interface requirements. While these requirements impose somewhat on model development, the benefits gained through the architecture far outweigh the imposition.

4.1 GNC Simulation Configuration (input data)

The minimum data set for the GNC simulation configuration structure is all the sensor, navigation, guidance, control output data structures and all of the effector model output forces and torques. Since the configuration structure has parameters and sub-structures of the same type as all the output structures and parameters, the "selected" output can be directed via a pointer assigned in the input file. If no selection is desired, this structure is not needed and the outputs are fed directly to the appropriate functions. Figure 7 contains an example of GNC simulation configuration input data pointer assignments.

```
/* Select state and status structures */
  cev_cm_veh.sim_config.guid_ss  = &cev_cm_veh.guid_ss;
//cev_cm_veh.sim_config.nav_ss   = &cev_cm_veh.hf_nav_ss;
  cev_cm_veh.sim_config.nav_ss   = &cev_cm_veh.p_nav_ss;
  cev_cm_veh.sim_config.cntrl_ss = &cev_cm_veh.cntrl_ss;

/* Select sensor models */
//cev_veh.sim_config.LDO = &cev_veh.lidar1.LDO;
  cev_veh.sim_config.LDO = &cev_veh.lidarN.LDO;
//cev_veh.sim_config.IMUO = &cev_veh.imu1.IMUO;
  cev_veh.sim_config.IMUO = &cev_veh.imuN.IMUO;
//cev_veh.sim_config.GPSO = &cev_veh.gps1.GPSO;
  cev_veh.sim_config.GPSO = &cev_veh.gpsN.GPSO;
//cev_veh.sim_config.STRO = &cev_veh.str1.STRO;
  cev_veh.sim_config.STRO = &cev_veh.strN.STRO;

/* Select navigation */
//cev_veh.sim_config.NAV = &cev_veh.p_nav;
//cev_veh.sim_config.NAV = &cev_veh.pseudo_nav;
  cev_veh.sim_config.NAV = &cev_veh.hf_nav;

/* Select guidance */
//cev_veh.sim_config.GO = &cev_veh.guid_out1;
  cev_veh.sim_config.GO = &cev_veh.guid_outN;

/* Select control */
//cev_veh.sim_config.CTRL = &cev_veh.cntrl_out1;
  cev_veh.sim_config.CTRL = &cev_veh.cntrl_outN;

/* select effector models */
//cev_veh.sim_config.eff_force  = &cev_veh.peff.force[0];
//cev_veh.sim_config.eff_torque = &cev_veh.peff.torque[0];
  cev_veh.sim_config.eff_force  = &cev_veh.rcs.force[0];
  cev_veh.sim_config.eff_torque = &cev_veh.rcs.torque[0];
```

Figure 7: GNC Simulation Configuration Input Data Example

4.2 Sensor Models (S_define calls)

The minimum data set required for sensor models is the sensor-related data structure and the generic output data structure appropriate for the measurement type. Sensor models will also likely have "truth" parameters as inputs for measurement modeling. Furthermore, each sensor model structure must have an "enable" flag that will be set via input data and/or have an "enable" flag in the calling argument list. Obviously, the sensor model logic must key off of the "enable" flag. Note, each type of sensor measurement must have a generic output data structure type accessible via the Common Model Library (CML). This output structure must be available at the S_define level as an input directly to the navigation routines. The minimum contents of this output structure are a valid-measurement flag (from the sensor hardware point of view), a time tag, and the measurement value. The output structure can either be a sub-structure of the sensor structure or instantiated separately in the S_define file. Figure 8 contains an example of a pose sensor (LIDAR) model interface.

```
lidar_model(
                        $$$$    /* Others as needed */
   RELKIN               * RKN,   /* IN:     --  True relative states */
   lidar_T              * LD,    /* INOUT:  --  LIDAR data struct */
   pose_out_T           * LDO)   /* OUT:    --  LIDAR output data struct */

or

lidar_model(
                        $$$$    /* Others as needed */
   RELKIN               * RKN,   /* IN:     --  True relative states */
   lidar_T              * LD)    /* INOUT:  --  LIDAR data struct */

where pose_out_T * LD0 = &(LD->LD0);
```

Figure 8: Sensor Model Interface Examples

4.3 Vehicle Executive (S_define call)

The minimum data set for the Vehicle Executive is all rate group frequencies (dt's), current time, general navigation data structure, GNC states and status data structures, and the vehicle executive data structure. Figure 9 contains an example of the Vehicle Executive interface.

```
veh_veh_executive(
                        $$$$   /* Others if needed */
   double          high_rate_dt, /* IN:     s   High rate time step        */
   double           med_rate_dt, /* IN:     s   Medium rate time step      */
   double                  time, /* IN:     s   Simulation time            */
   general_nav_data_T     * NAV, /* IN:     --  General nav data structure */
   veh_gnc_guid_ss_T      * SSG, /* IN:     --  Guidance state and status  */
   veh_gnc_nav_ss_T       * SSN, /* IN:     --  Nav state and status       */
   veh_gnc_cntrl_ss_T     * SSC, /* IN:     --  Control State and status   */
   veh_veh_executive_T    * VEH) /* INOUT:  --  Vehicle Exec data struc    */
```

Figure 9: Vehicle Executive Interface Example

4.4 Pseudo GNC Executive (S_define calls)

The minimum data set for the Pseudo Pre GNC Executive is an enable flag and the vehicle executive data structure. The minimum data set for the Pseudo Post GNC Executive is an enable flag, the GNC state and status data structures, and the vehicle executive data structure. Figure 10 contains an example of the Pseudo Executive interfaces.

```
veh_high_rate_pseudo_pre_gnc_exec(
                        $$$$   /* Others if needed  */
   Flag                 enable,/* IN:     --  Enable flag                */
   veh_veh_executive_T *VEH)  /* INOUT:  --  Vehicle Exec data struct  */


veh_high_rate_pseudo_post_gnc_exec(
                        $$$$   /* Others if needed  */
   Flag                 enable,/* IN:     --  Enable flag                */
   veh_veh_executive_T *VEH,  /* INOUT:  --  Vehicle Exec data struct  */
   veh_gnc_guid_ss_T    * SSG, /* IN:     --  SC->guid_ss  (sim_config) */
   veh_gnc_nav_ss_T     * SSN, /* IN:     --  SC->nav_ss   (sim_config) */
   veh_gnc_cntrl_ss_T   * SSC) /* IN:     --  SC->cntrl_ss (sim_config) */
```

Figure 10: Pseudo GNC Executive Interface Examples

4.5 Navigation

4.5.1 Manager Function (S_define call)

The minimum data set for the Navigation Manager is an “enable” flag, all the required sensor output data structures, the navigation system data structure (truth data for perfect navigation), the general navigation data structure, navigation state and status data structure, and the vehicle executive data structure. The Navigation Manager may bring in other parameters as needed, but that is all that is required for the architecture. The “enable” flag may be set via a local parameter declared in the S_define file (set in the input file). The sensor output structures are needed for filter measurements (not needed for perfect or pseudo/simple nav_mngr); the navigation system

structure is needed for model-dependent parameters; the general navigation structure is used to housed all necessary navigation parameters for guidance and control; the state and status structure is used to pass additional information to the Vehicle Executive for mode transitions; and the vehicle executive structure is needed for Phase, Segment, Mode, and Sub-mode identification. Figure 11 contains an example of the Navigation Manager interface (high rate).

```
veh_high_rate_hifi_nav_mngr(
                        $$$$     /* Others as needed */
   Flag                 enable,  /* IN:    --  hifi_nav_enable               */
   pose_out_T           * LDO,   /* IN:    --  LIDAR output                  */
   pose_out_T           * NFO,   /* IN:    --  NFIR output                   */
   imu_out_T            * IMUO,  /* IN:    --  IMU output                    */
   gps_out_T            * GPSO,  /* IN:    --  GPS output                   */
   str_out_T            * STRO,  /* IN:    --  Star trckr out               */
   veh_hifi_nav_T       * HFN,   /* INOUT: --  Hi-fi navigation struct       */
   veh_veh_executive_T  * VEH,   /* INOUT: --  Vehicle Exec data struct      */
   general_nav_T        * NAV,   /* OUT:   --  General navigation struct      */
   veh_gnc_nav_ss_T     * SSN)   /* OUT:   --  Nav state and status          */
```

Figure 11: Navigation Manager Interface Example

4.5.2 Navigation and UPP Functions (called by Navigation Manager)

The only interface restrictions on the navigation functions are that the measurement data used must come from one of the available sensor output structures and that the necessary parameters in the general navigation structure be assigned. The general navigation assignments can be done within the navigation functions or in the Navigation Manger. When possible, the general navigation parameters should be computed using generic universal parameter processor (UPP) type functions. Figure 12 contains examples of some hi-fi navigation function interfaces.

```
veh_nav_input(
                        $$$$    /* Others as needed */
   double               time,   /* IN:    s   Current time        */
   pose_out_T         * LDO,    /* IN:    --  LIDAR output         */
   pose_out_T         * NFO,    /* IN:    --  NFIR output          */
   imu_out_T          * IMUO,   /* IN:    --  IMU output           */
   gps_out_T          * GPSO,   /* IN:    --  GPS output           */
   str_out_T          * STRO,   /* IN:    --  Star trckr out       */
   GKF_Obs            * G)      /* INOUT: --  Filter observations */

veh_hifi_nav(
                        $$$$    /* Others as needed */
   double               time,   /* IN:    s   Current time               */
   GKF_Obs            * G,      /* INOUT: --  Filter observations        */
   general_nav_T      * NAV)    /* OUT:   --  General navigation struct */
   veh_hifi_nav_T     * HFN)    /* INOUT: --  Hi-fi navigation struct   */

abs_upp(
   veh_hifi_nav_T     * HFN,    /* IN:    --  Hi-fi navigation struct   */
   general_nav_T      * NAV)    /* OUT:   --  General navigation struct */

rel_upp(
   veh_hifi_nav_T     * HFN,    /* IN:    --  Hi-fi navigation struct   */
   general_nav_T      * NAV)    /* OUT:   --  General navigation struct */
```

Figure 12: Navigation and UPP Function Interface Examples

4.6 Guidance

4.6.1 Manager Function (S_define call)

The minimum data set for the Guidance Manager is an “enable” flag, the general navigation structure, the necessary guidance structure(s), the general guidance output structure, guidance state and status data structure, and the vehicle executive data structure. The Guidance Manager may bring other parameters as needed, but that is all that is required for the architecture. The “enable” flag will be set via a local parameter declared in the S_define file (set in the input file). The general navigation structure is needed for guidance inputs; the guidance structure(s) is needed for model-dependent parameters; the general guidance output structure is needed for inputs to control; the state and status structure is used to pass additional information to the Vehicle Executive for mode transitions; and the vehicle executive structure is needed for Phase, Segment, Mode, and Sub-mode identification. Note, the general guidance output structure must include, but is not limited to: the commanded force/dv, the commanded and errors in position, velocity, attitude, and rate. Figure 13 contains an example of a Guidance Manager interface (high rate).

```
cev_high_rate_guid_mngr(
                        $$$$   /* Others as needed */
   Flag                   enable,   /* IN:     --  guidance enable              */
   general_nav_T          * NAV,    /* IN:     --  (SC->NAV/NAV) (sim_config)   */
   veh_specific_guid_T    * VG,     /* INOUT: --  Vehicle guid structures       */
   veh_veh_executive_T    * VEH,    /* INOUT: --  Vehicle Exec data struct      */
   general_guid_out_T     * GO,     /* OUT:    --  General guidance output       */
   veh_gnc_guid_ss_T      * SSG)    /* OUT:    --  Guidance state and status     */
```

Figure 13: Guidance Manager Interface Example

4.6.2 Guidance Routines (called by Guidance Manager)

The only interface restrictions on the guidance functions are that the navigation data used must come from the general navigation structure and that the necessary parameters in the general guidance output structure be assigned. The general guidance output assignments can be done within the guidance functions or in the Guidance. Figure 14 contains examples of a translational and rotational guidance function interface respectively. Note, in the translational guidance example, the mapping to the general output data structure would be done in the Manager function.

```
cw_bcb_maneuvers(
   double time,             /* IN:     s    Simulation time                  */
   double mass_est,         /* IN:     kg   Vehicle Mass                     */
   double mean_motion,      /* IN:     r/s  (NAV->….orb_rate)                 */
   double rel_pos_tlvlh[3],/* IN:     M    (NAV->…. R_tcom_ccom_tlvch)       */
   double rel_vel_tlvlh[3],/* IN:     M/s  (NAV->…. V_tcom_ccom_tlvlh_tlvch) */
   CW_BURN_C_BURN * BCB)    /* INOUT:  --   Burn coast burn data              */

cml_att_rate_guid(
   double             time, /* IN:     s    Simulation time            */
   INAV               *  N, /* IN:     --   Chaser nav (NAV->…)        */
   INAV               * TN, /* IN:     --   Target nav (NAV->…)        */
   RNAV               * RN, /* IN:     --   Relative nav (NAV->…       */
   ATT_RATE_GUID      *  G, /* INOUT:  --   Att/rate guidance data     */
   general_guid_out_T * GO) /* OUT:    --   General guidance output    */
```

Figure 14: Guidance Function Interface Examples

4.7 Control

4.7.1 Manager Function (S_define call)

The minimum data set for the Control Manager is an “enable” flag, the general guidance output structure, the general navigation structure, the necessary control structure(s), the general control structure, the necessary jet selection structure(s), control state and status data structure, and the vehicle executive data structure. The Control Manager may bring other parameters as needed,

but that is all that is required for the architecture. The "enable" flag will be set via a local parameter declared in the S_define file (set in the input file). The general guidance output structure is needed for control inputs; the general navigation structure is needed for coordinate transformations, feed-forward commands, and other miscellaneous computations; the control structure(s) is needed for model-dependent parameters; the general control output structure is needed for inputs to the effector routines; the jet selection structure(s) is needed for model-dependent parameters; the state and status structure is used to pass additional information to the Vehicle Executive for mode transitions; and the vehicle executive structure is needed for Phase, Segment, Mode, and Sub-mode identification. Note, the general control output structure must include, but is not limited to: commanded force and torque, and jet commands. Figure 15 contains an example of a Control Manager interface (high rate).

```
cev_high_rate_control_mngr(
                        $$$$   /* Others as needed */
   Flag                   enable,  /* IN:    --  c1_enable                  */
   general_guid_out_T     * GO,    /* IN:    --  (SC->GO/GO) (sim_config)   */
   general_nav_T          * NAV,   /* IN:    --  (SC->NAV/NAV) (sim_config) */
   CEV_CONTROL            * CON,   /* INOUT: --  Veh control struct          */
   CEV_JET_SELECT         * JS,    /* INOUT: --  Veh jet selection struct    */
   veh_veh_executive_T    * VEH,   /* INOUT: --  Vehicle Exec data struct    */
   general_control_out_T  * CO,    /* OUT:   --  Control guidance output     */
   veh_gnc_cntrl_ss_T     * SSC)   /* OUT:   --  Control state and status    */
```

Figure 15: Control Manager Interface Example

4.7.2 Control, Saturation, & Jet Selection Functions (called by Control Manager)

The only interface restrictions on these functions are that the guidance data used must come from the general guidance output structure, the navigation data used must come from the general navigation structure, the necessary parameters in the general control structure must be assigned (if applicable). The general control output assignments can be done within these functions or in the Control Manger. Figure 16 contains examples of translational and rotational control, command saturation, and jet selection function interfaces respectively. Note, in the following examples, the mapping to the general control output data structure would be done in the Manager function.

```
pd_pos_con (
   double pos_err[3],          /* IN: -- (GO->pos_err)                */
   double vel_err[3],          /* IN: -- (GO->vel_err)                */
   double T_body_cmdfrm[3][3], /* IN: -- (NAV->…. T_cbody_tlvlh        */
   double m_est,               /* IN: kg estimated vehicle mass       */
   PD_POS_CON * POS)           /* INOUT: -- &(CON->CN2)               */

pd_att_con(
   double        q_err[4],     /* IN:     --   (GO->q_err)             */
   double        rate_err[3],  /* IN:     --   (GO->rate_err)          */
   double        T_dist[3],    /* IN:     --   disturbance torque      */
   double        I_est[3],     /* IN:     kgM2 estimated axial inertia */
   PD_ATT_CON    * ATT)        /* INOUT:  --   &(CON->CN1)             */

simplex(
  double      dv[3],    /* IN: M/s delta-v trans command (from CO param)   */
  double      dw[3],    /* IN: r/s delta-omega rot command (from CO param) */
  RCS_SIMPLEX *simplex) /* INOUT:  -- simplex data structure               */
```

Figure 16: Control, Saturation, & Jet Selection Function Interface Examples

4.8 Effector Functions (S_define calls)

The only interface restrictions on the effector functions are that the jet selection data (control data for perfect effectors) used must come from the general control output structure, and that the output force and torque must be in the effector data structure. Figure 17 contains an example of a simple effector and an rcs function interface respectively.

```
dvdw_FT_Imp(
  double       * c_dv,          /* INOUT:  M/s  Commanded dv (NULL)          */
  double       * c_dw,          /* INOUT:  r/s  Commanded dw (NULL)          */
  double       * c_F,           /* INOUT:  N    (SC->CO.control_force)       */
  double       * dt_force,      /* INOUT   s    Desired appl time for Force  */
  double       * c_T,           /* INOUT:  NM   (SC->CO.control_torque)      */
  double       * dt_torque,     /* INOUT   s    Desired appl time for Torque */
  double       * c_lin_imp,     /* INOUT   Ns   Cmnd Linear Impulse (NULL)   */
  double       * c_ang_imp,     /* INOUT   NMs  Cmnd Angular Impulse (NULL)  */
  double         T_L_I[3][3],   /* IN:     --   LVLH to INRTL Rot           */
  OrbitalBody  * ORB,           /* IN:     --   Veh orbital bdy struc       */
  DynMassGroup * massgrp,       /* OUT:    --   DynMass struc for veh       */
  DVDW_FT_IMP  * PF)            /* INOUT:  --   Perfect effector struct     */

rcs_generic(
   RCS_GENERIC    * R),     /* Inout:  -- (rcs)                              */
   DynMassGroup   * M ,     /* Inout:  -- The dynamic mass group structure   */
   MassProperties * P ,     /* In:     -- Composite vehicle mass properties  */
   int    *rcs_command,     /* In:     -- (SC->CO[0].jet_cmnds)              */
```

Figure 17: Effector Model Interface Examples

4.9 Trick Collect Mechanism (Statements in S_define)

Figure 18 contains an example of a trick collect mechanism interface for force and torque.

```
/* COLLECT EFFECTOR FORCES FOR VEHICLE */
collect veh.body.orbit.trans.collect_effector_forc =
{
   veh.gnc_sim_config.eff_force[0]
};

/* COLLECT EFFECTOR TORQUES FOR VEHICLE */
collect veh.body.orbit.rot.collect_effector_torq =
{
   veh.gnc_sim_config.eff_torque[0]
};
```

Figure 18: Effector & Torque Trick Collection Interface Examples

Appendix A: General Vehicle Executive Actions

♦ Change Phase, Segment, and or Task Immediately (aborts etc.)

```
typedef struct { /* ---------------------------------- cev_veh_executive_T */
   Flag override_phase;            /* --  Yes to override phase (now)     */
   Flag override_segment;          /* --  Yes to override segment (now)   */
   Flag override_task;             /* --  Yes to override task (now)      */
   Flag override_task_completion;/* --  Yes to override task completion(now)*/
```

♦ Specify New Phase, Segment, and or Task Upon Nominal Task Completion

(These commands are used when the nominal task-incrementing is NOT desired.)

```
typedef struct { /* ---------------------------------------- cev_veh_task_T */
   Flag goto_phase;         /* -- Yes to jump to specified flight phase upon
                               task completion                              */
   Flag goto_segment;       /* -- Yes to jump to specified flight segment
                               upon task completion                         */
   Flag goto_task;          /* -- Yes to jump to specified task upon
                               task completion                              */
```

♦ Repeat a Set of Tasks or Skip the Next Task

```
typedef struct { /* ---------------------------------------- cev_veh_task_T */
   Flag repeat_tasks;    /* -- Yes/No to repeat a set of tasks             */
   Flag skip_next_task;  /* -- Yes/No to skip the next task                */
```

♦ Optional "Go" (authority to proceed) for Mode Transition

```
typedef struct { /* ---------------------------------------- cev_veh_task_T */
   Flag go_needed;       /* -- Yes/No on needing a "go" for next task      */
   Flag go;              /* -- Yes/No for task execution (if needed)       */
```

♦ Update Task Data

```
typedef struct { /* ---------------------------------------- cev_veh_task_T */
   Flag  update_t_guid_data;    /* -- Yes/No to update trans guid task data */
   Flag  update_a_guid_data;    /* -- Yes/No to update att guid task data   */
   Flag  update_t_nav_data;     /* -- Yes/No to update trans nav task data  */
   Flag  update_a_nav_data;     /* -- Yes/No to update att nav task data    */
   Flag  update_t_cntrl_data;   /* -- Yes/No to update trans cntrl task data*/
   Flag  update_a_cntrl_data;   /* -- Yes/No to update att cntrl task data  */
```